Russell's Metatheoretic Study - Notes

Matthieu Sozeau
LRI - Paris Sud - XI University
sozeau@lri.fr

16th July 2006

Abstract

We are working on the formalization of Russell's type theory in the Coq proof assistant [3]. The type system of Russell is based on the Calculus of Constructions with Σ-types (dependent sums), extended by an equivalence on types which subsumes β-conversion. The extension permits to identify types and subsets based on them in a manner similar to the *Predicate Subtyping* feature of PVS.

We are aiming at a complete proof of Russell's metatheoretic properties (structural properties, Subject Reduction, maybe Strong Normalization), the refining steps which led us to the algorithmic system and the corresponding typing algorithm and also the correctness of an interpretation from Russell to the Calculus of Inductive Constructions with metavariables.

We started the development using the formalization of the Calculus of Constructions by Bruno Barras [2]. We kept the standard de Bruijn encoding for variable bindings and defined our judgements using *dependent* inductive predicates. This alone causes some problems for the faithful formalization of the paper results. The proofs offer several other technical difficulties including:

- Elimination of transitivity in a system with an *untyped* type conversion relation.
- Subject Reduction, which is *not* directly provable for the declarative system. Here we adapted the technique developed by Robin Adams [1]. It includes a new term algebra, with associated reduction operations and a new type system for which we have to prove metatheoretic properties.
- Correction of the interpretation: the target system include metavariables, introducing a second, *unusual* kind of variable binding.

Organization of the Coq proof

The proof is developed in the `Lambda` namespace. At the root we have definition of the term language `Lambda.Terms` which is a lambda calculus with dependent products, sums and a distinguished subset type with no introduction or elimination constructs. In `LiftSubst` we have lemmas on substitution and lifting. The definitions and proofs about reduction and conversion including Church-Rosser for $\beta\pi$ reduction are in the remaining modules.

The first type system is `Lambda.CCSum`, the Calculus of Constructions enriched with dependent sums. All the metatheory up to Subject Reduction and Unicity has been done for this system, simply adapting the work of Bruno Barras.

Then we have two versions of the Russell type system. The `Lambda.JRussell` directory contains a definition of Russell with judgemental equality and a coercion relation with transitivity

and symetry built-in. It is the most declarative, and thus the clearest presentation of RUSSELL, however we can't prove Subject Reduction for this system directly. We proved basic metatheory for this system up to validity and unicity of sorting, which we can transpose to TPOSR easily.

The modules in `Lambda.Russell` define a more algorithmical version of the RUSSELL type system. This version uses an untyped conversion relation and an already refined coercion algorithm (without transitivity).

Finally, one can find in `Lambda.Meta` modules the metatheorems about those systems, or more precisely the equivalence proofs.

The proof of Subject Reduction

Directly proving Subject Reduction for our system is not possible because of the enriched equivalence we use. Indeed, we use a typed equivalence relation, hence we cannot use the same style of proof as the one for the Calculus of Constructions. In the later case, when proving subject reduction at the application case, with reduction being β, we have:

$$\frac{\Gamma \vdash (\lambda x : A.M) : \Pi x : B.C \qquad \Gamma \vdash N : B}{\Gamma \vdash (\lambda x : A.M)\ N : C[N/x]}$$

We need to prove $\Gamma \vdash M[N/x] : C[N/x]$.

By inversion for dependent products, we have $(s, s') \in \mathcal{A}$ so that

$$\begin{aligned} \Gamma &\vdash A : s \\ \Gamma, x : A &\vdash D : s' \\ \Gamma, x : A &\vdash M : D \end{aligned}$$

and $\Pi x : B.C \equiv_{\beta\pi} \Pi x : A.D$ ($\Gamma \vdash \Pi x : B.C = \Pi x : A.D : s'$ for typed equivalence). By the Church-Rosser theorem for $\equiv_{\beta\pi}$ we get $A \equiv_{\beta\pi} B$ and $C \equiv_{\beta\pi} D$. Hence we have $\Gamma \vdash N : A$ and $\Gamma, x : A \vdash M : C$ by conversion. By substitution we can then derive $\Gamma \vdash M[N/x] : C[N/x]$.

What goes wrong in the typed version ? We would need to prove the following lemma (injectivity of pi):

$$\Gamma \vdash \Pi x : A.B = \Pi x : C.D : s' \Rightarrow \exists s, \Gamma \vdash A = C : s \wedge \Gamma, x : A \vdash B = D : s \wedge (s, s') \in \mathcal{A}$$

However, this is not provable as we can use a transitivity rule in the definition of typed equivalence. Here the premise can have the form $\Gamma \vdash \Pi x : A.B = X_0 \wedge \ldots \wedge X_n = \Pi x : C.D : s$ and we can't say anything about the X's yet, in particular we cannot prove that they reduce to products. Hence we need to devise a new, equivalent system for which subject reduction is provable. We can do that using a typed parallel one step reduction relation (TPOSR) on *labelled* terms which generates the typed equivalence relation on the original terms. The formalization as a reduction relation, along with its proof of Church-Rosser eliminates transitivity and permits to prove Uniqueness of Types and Injectivity of products (and sums in our case). It is then possible to prove Subject Reduction for this system and transpose this result to the judgemental equality system by virtue of the equivalence of the two systems.

Proof sketch

The technique of using TPOSR is due to Robin Adams work on proving the equivalence of judgemental equality (or typed equivalence) and untyped conversion presentations of Pure Type Systems [1]. We extended this technique to an enriched equivalence relation which we call coercion (denoted by $\rhd$). Coercion includes the usual equivalence relation generated by the reduction rules of the system (β and π in our case) and extends it with custom equality rules. In the RUSSELL system, we added the following rules:

$$\rhd\text{-PROOF}\ \frac{\Gamma \vdash U \rhd V : \texttt{Set} \qquad \Gamma, x : V \vdash P : \texttt{Prop}}{\Gamma \vdash U \rhd \{\ x : V \mid P\ \} : \texttt{Set}}$$

$$\rhd\text{-SUBSET}\ \frac{\Gamma \vdash U \rhd V : \texttt{Set} \qquad \Gamma, x : U \vdash P : \texttt{Prop}}{\Gamma \vdash \{\ x : U \mid P\ \} \rhd V : \texttt{Set}}$$

The following module is the definition of the RUSSELL system with judgemental equality in COQ.

Reserved Notation "$G \vdash T = U : s$" (*at level* 70, *T*, *U*, *s at next level*).
Reserved Notation "$G \vdash T \rhd U : s$" (*at level* 70, *T*, *U*, *s at next level*).
Reserved Notation "$G \vdash T : U$" (*at level* 70, *T*, *U at next level*).

Definition *sum_sort s1 s2 s3* :=
$(s1 = set \wedge s2 = set \wedge s3 = set) \vee$
$(s1 = prop \wedge s2 = prop \wedge s3 = prop)$.

Inductive *coerce* : *env* → *term* → *term* → *sort* → *Prop* :=
| *coerce_conv* : ∀ *e A B s*, $e \vdash A = B : Srt\ s \rightarrow e \vdash A \rhd B : s$

| *coerce_weak* : ∀ *e A B s*, $e \vdash A \rhd B : s \rightarrow$
∀ *T s'*, $e \vdash T : Srt\ s' \rightarrow (T :: e) \vdash lift\ 1\ A \rhd lift\ 1\ B : s$

| *coerce_prod* : ∀ *e A B A' B'*,
∀ *s*, $e \vdash A' \rhd A : s \rightarrow$
$e \vdash A' : Srt\ s \rightarrow e \vdash A : Srt\ s \rightarrow$
∀ *s'*, $(A' :: e) \vdash B \rhd B' : s' \rightarrow$
$A :: e \vdash B : Srt\ s' \rightarrow A' :: e \vdash B' : Srt\ s' \rightarrow$
$e \vdash (Prod\ A\ B) \rhd (Prod\ A'\ B') : s'$

| *coerce_sum* : ∀ *e A B A' B'*,
∀ *s*, $e \vdash A \rhd A' : s \rightarrow$
$e \vdash A' : Srt\ s \rightarrow e \vdash A : Srt\ s \rightarrow$
∀ *s'*, $(A :: e) \vdash B \rhd B' : s' \rightarrow$
$A :: e \vdash B : Srt\ s' \rightarrow A' :: e \vdash B' : Srt\ s' \rightarrow$
∀ *s"*, *sum_sort s s' s"* → *sum_sort s s' s"* →

$e \vdash (Sum\ A\ B) \rhd (Sum\ A'\ B') : s$"

| $coerce\_sub\_l : \forall\ e\ U\ P\ U'$,
$e \vdash U \rhd U' : set \to$
$e \vdash U : Srt\ set \to e \vdash U' : Srt\ set \to$
$U :: e \vdash P : Srt\ prop \to$
$e \vdash Subset\ U\ P \rhd U' : set$

| $coerce\_sub\_r : \forall\ e\ U\ U'\ P$,
$e \vdash U \rhd U' : set \to$
$e \vdash U : Srt\ set \to e \vdash U' : Srt\ set \to$
$U' :: e \vdash P : Srt\ prop \to$
$e \vdash U \rhd (Subset\ U'\ P) : set$

| $coerce\_sym : \forall\ e\ U\ V\ s, e \vdash U \rhd V : s \to e \vdash V \rhd U : s$

| $coerce\_trans : \forall\ e\ A\ B\ C\ s$,
$e \vdash A \rhd B : s \to e \vdash B \rhd C : s \to e \vdash A \rhd C : s$

$where\ "G \vdash T \rhd U : s" := (coerce\ G\ T\ U\ s)$

$with\ jeq : env \to term \to term \to term \to Prop :=$
| $jeq\_weak : \forall\ e\ M\ N\ A, e \vdash M = N : A \to$
$\forall\ B\ s, e \vdash B : Srt\ s \to$
$(B :: e) \vdash lift\ 1\ M = lift\ 1\ N : lift\ 1\ A$

| $jeq\_prod : \forall\ e\ U\ V\ U'\ V'\ s1\ s2$,
$e \vdash U = U' : Srt\ s1 \to U :: e \vdash V = V' : Srt\ s2 \to$
$e \vdash Prod\ U\ V = Prod\ U'\ V' : Srt\ s2$

| $jeq\_abs : \forall\ e\ A\ A'\ s1, e \vdash A = A' : Srt\ s1 \to$
$\forall\ B\ s2, (A :: e) \vdash B : Srt\ s2 \to$
$\forall\ M\ M', (A :: e) \vdash M = M' : B \to$
$e \vdash Abs\ A\ M = Abs\ A'\ M' : (Prod\ A\ B)$

| $jeq\_app : \forall\ e\ A\ B\ M\ M', e \vdash M = M' : (Prod\ A\ B) \to$
$\forall\ N\ N', e \vdash N = N' : A \to$
$e \vdash App\ M\ N = App\ M'\ N' : subst\ N\ B$

| $jeq\_beta : \forall\ e\ A\ s1, e \vdash A : Srt\ s1 \to$
$\forall\ B\ s2, (A :: e) \vdash B : Srt\ s2 \to$
$\forall\ M, (A :: e) \vdash M : B \to$
$\forall\ N, e \vdash N : A \to$
$e \vdash App\ (Abs\ A\ M)\ N = subst\ N\ M : subst\ N\ B$

$|\ jeq\_sum : \forall\ e\ A\ A'\ s1,\ e \vdash A = A' : Srt\ s1 \rightarrow$
$\forall\ B\ B'\ s2,\ (A :: e) \vdash B = B' : Srt\ s2 \rightarrow$
$\forall\ s3,\ sum\_sort\ s1\ s2\ s3 \rightarrow$
$e \vdash Sum\ A\ B = Sum\ A'\ B' : Srt\ s3$

$|\ jeq\_pair : \forall\ e\ A\ A'\ s1,\ e \vdash A = A' : Srt\ s1 \rightarrow$
$\forall\ B\ B'\ s2,\ (A :: e) \vdash B = B' : Srt\ s2 \rightarrow$
$\forall\ s3,\ sum\_sort\ s1\ s2\ s3 \rightarrow$
$\forall\ u\ u',\ e \vdash u = u' : A \rightarrow$
$\forall\ v\ v',\ e \vdash v = v' : subst\ u\ B \rightarrow$
$e \vdash Pair\ (Sum\ A\ B)\ u\ v = Pair\ (Sum\ A'\ B')\ u'\ v' : Sum\ A\ B$

$|\ jeq\_pi1 : \forall\ e\ t\ t'\ A\ B,\ e \vdash t = t' : Sum\ A\ B \rightarrow$
$e \vdash Pi1\ t = Pi1\ t' : A$

$|\ jeq\_pi1\_red : \forall\ e\ A\ s1,\ e \vdash A : Srt\ s1 \rightarrow$
$\forall\ B\ s2,\ (A :: e) \vdash B : Srt\ s2 \rightarrow$
$\forall\ s3,\ sum\_sort\ s1\ s2\ s3 \rightarrow$
$\forall\ u,\ e \vdash u : A \rightarrow \forall\ v,\ e \vdash v : subst\ u\ B \rightarrow$
$e \vdash Pi1\ (Pair\ (Sum\ A\ B)\ u\ v) = u : A$

$|\ jeq\_pi2 : \forall\ e\ t\ t'\ A\ B,\ e \vdash t = t' : Sum\ A\ B \rightarrow$
$e \vdash Pi2\ t = Pi2\ t' : subst\ (Pi1\ t)\ B$

$|\ jeq\_pi2\_red : \forall\ e\ A\ s1,\ e \vdash A : Srt\ s1 \rightarrow$
$\forall\ B\ s2,\ (A :: e) \vdash B : Srt\ s2 \rightarrow$
$\forall\ s3,\ sum\_sort\ s1\ s2\ s3 \rightarrow$
$\forall\ u,\ e \vdash u : A \rightarrow \forall\ v,\ e \vdash v : subst\ u\ B \rightarrow$
$e \vdash Pi2\ (Pair\ (Sum\ A\ B)\ u\ v) = v : subst\ u\ B$

$|\ jeq\_subset : \forall\ e\ A\ A',\ e \vdash A = A' : Srt\ set \rightarrow$
$\forall\ B\ B',\ (A :: e) \vdash B = B' : Srt\ prop \rightarrow$
$e \vdash Subset\ A\ B = Subset\ A'\ B' : Srt\ set$

$|\ jeq\_refl : \forall\ e\ M\ A,\ e \vdash M : A \rightarrow e \vdash M = M : A$

$|\ jeq\_sym : \forall\ e\ M\ N\ A,\ e \vdash M = N : A \rightarrow e \vdash N = M : A$

$|\ jeq\_trans : \forall\ e\ M\ N\ P\ A,\ e \vdash M = N : A \rightarrow e \vdash N = P : A \rightarrow e \vdash M = P : A$

$|\ jeq\_conv : \forall\ e\ M\ N\ A\ B\ s,\ e \vdash M = N : A \rightarrow e \vdash A \rhd B : s \rightarrow e \vdash M = N : B$

$where\ "G \vdash T = U : s" := (jeq\ G\ T\ U\ s)$

$with\ typ : env \rightarrow term \rightarrow term \rightarrow Prop :=$

$|\ type\_prop : nil \vdash (Srt\ prop) : (Srt\ kind)$
$|\ type\_set : nil \vdash (Srt\ set) : (Srt\ kind)$
$|\ type\_var :$
$\forall\ e\ T\ s,\ e \vdash T : Srt\ s \rightarrow (T :: e) \vdash (Ref\ 0) : (lift\ 1\ T)$
$|\ type\_weak :$
$\forall\ e\ t\ T,\ e \vdash t : T \rightarrow \forall\ U\ s,\ e \vdash U : Srt\ s \rightarrow$
$(U :: e) \vdash lift\ 1\ t : lift\ 1\ T$
$|\ type\_abs :$
$\forall\ e\ T\ s1,$
$e \vdash T : (Srt\ s1) \rightarrow$
$\forall\ M\ (U : term)\ s2,$
$(T :: e) \vdash U : (Srt\ s2) \rightarrow$
$(T :: e) \vdash M : U \rightarrow$
$e \vdash (Abs\ T\ M) : (Prod\ T\ U)$
$|\ type\_app :$
$\forall\ e\ v\ (V : term),\ e \vdash v : V \rightarrow$
$\forall\ u\ (Ur : term),\ e \vdash u : (Prod\ V\ Ur) \rightarrow$
$e \vdash (App\ u\ v) : (subst\ v\ Ur)$

$|\ type\_pair :$
$\forall\ e\ (U : term)\ s1,\ e \vdash U : (Srt\ s1) \rightarrow$
$\forall\ u,\ e \vdash u : U \rightarrow$
$\forall\ V\ s2,\ (U :: e) \vdash V : (Srt\ s2) \rightarrow$
$\forall\ v,\ e \vdash v : (subst\ u\ V) \rightarrow$
$\forall\ s3,\ sum\_sort\ s1\ s2\ s3 \rightarrow$
$e \vdash (Pair\ (Sum\ U\ V)\ u\ v) : (Sum\ U\ V)$

$|\ type\_prod :$
$\forall\ e\ T\ s1,$
$e \vdash T : (Srt\ s1) \rightarrow$
$\forall\ (U : term)\ s2,$
$(T :: e) \vdash U : (Srt\ s2) \rightarrow$
$e \vdash (Prod\ T\ U) : (Srt\ s2)$

$|\ type\_sum :$
$\forall\ e\ T\ s1,$
$e \vdash T : (Srt\ s1) \rightarrow$
$\forall\ (U : term)\ s2,$
$(T :: e) \vdash U : (Srt\ s2) \rightarrow$
$\forall\ s3,\ sum\_sort\ s1\ s2\ s3 \rightarrow$
$e \vdash (Sum\ T\ U) : Srt\ s3$

$|\ type\_subset :$
$\forall\ e\ T,\ e \vdash T : (Srt\ set) \rightarrow$
$\forall\ (U : term),\ (T :: e) \vdash U : (Srt\ prop) \rightarrow$

$e \vdash (Subset\ T\ U) : (Srt\ set)$

| *type_pi1* :
$\forall\ e\ t\ U\ V,\ e \vdash t : (Sum\ U\ V) \rightarrow$
$e \vdash (Pi1\ t) : U$

| *type_pi2* :
$\forall\ e\ t\ U\ V,\ e \vdash t : (Sum\ U\ V) \rightarrow$
$e \vdash (Pi2\ t) : (subst\ (Pi1\ t)\ V)$

| *type_conv* :
$\forall\ e\ t\ (U\ V : term),$
$e \vdash t : U \rightarrow$
$\forall\ s,\ e \vdash U \backsimeq V : s \rightarrow$
$e \vdash t : V$

where "$G \vdash T : U$" := $(typ\ G\ T\ U)$.

The proof works as follows:

TPOSR system

We define a TPOSR system on labelled terms. Here follows the corresponding Coq definitions (`Lambda.TPOSR.Types`).

Reserved Notation "$G \vdash T \rightarrow U : s$" (*at level* 70, T, U, s *at next level*).

Definition *sum_sort s1 s2 s3* :=
$(s1 = set \wedge s2 = set \wedge s3 = set) \vee$
$(s1 = prop \wedge s2 = prop \wedge s3 = prop)$.

Coercion Srt_l : *sort* > − > *lterm*.

Reserved Notation "$G \vdash T \simeq U : s$" (*at level* 70, T, U, s *at next level*).
Reserved Notation "$G \vdash T \backsimeq U : s$" (*at level* 70, T, U, s *at next level*).

Inductive *tposr_wf* : *lenv* → *Prop* :=
| *wf_nil* : *tposr_wf nil*
| *wf_cons* : $\forall\ G\ A\ s,\ G \vdash A \rightarrow A : s \rightarrow$ *tposr_wf* $(A :: G)$

with tposr : *lenv* → *lterm* → *lterm* → *lterm* → *Prop* :=

$|\ tposr\_var : \forall\ e,\ tposr\_wf\ e \rightarrow$
$\forall\ n\ T,\ item\_llift\ T\ e\ n \rightarrow e \vdash (Ref\_l\ n) \rightarrow (Ref\_l\ n) :\ T$

$|\ tposr\_set : \forall\ e,\ tposr\_wf\ e \rightarrow e \vdash set \rightarrow set : kind$

$|\ tposr\_prop : \forall\ e,\ tposr\_wf\ e \rightarrow e \vdash prop \rightarrow prop : kind$

$|\ tposr\_prod : \forall\ e\ A\ A'\ s1,\ e \vdash A \rightarrow A' : s1 \rightarrow$
$\forall\ B\ B'\ s2,\ (A :: e) \vdash B \rightarrow B' : s2 \rightarrow$
$e \vdash Prod\_l\ A\ B \rightarrow Prod\_l\ A'\ B' : s2$

$|\ tposr\_abs : \forall\ e\ A\ A'\ s1,\ e \vdash A \rightarrow A' : s1 \rightarrow$
$\forall\ B\ B'\ s2,\ (A :: e) \vdash B \rightarrow B' : s2 \rightarrow$
$\forall\ M\ M',\ (A :: e) \vdash M \rightarrow M' : B \rightarrow$
$e \vdash Abs\_l\ A\ M \rightarrow Abs\_l\ A'\ M' : (Prod\_l\ A\ B)$

$|\ tposr\_app : \forall\ e\ A\ A'\ s1,\ e \vdash A \rightarrow A' : s1 \rightarrow$
$\forall\ B\ B'\ s2,\ (A :: e) \vdash B \rhd B' : s2 \rightarrow$
$\forall\ M\ M',\ e \vdash M \rightarrow M' : (Prod\_l\ A\ B) \rightarrow$
$\forall\ N\ N',\ e \vdash N \rightarrow N' : A \rightarrow$
$e \vdash App\_l\ B\ M\ N \rightarrow App\_l\ B'\ M'\ N' : lsubst\ N\ B$

$|\ tposr\_beta : \forall\ e\ A\ A'\ s1,\ e \vdash A \rightarrow A' : s1 \rightarrow$
$\forall\ B\ B'\ s2,\ (A :: e) \vdash B \rightarrow B' : s2 \rightarrow$
$\forall\ M\ M',\ (A :: e) \vdash M \rightarrow M' : B \rightarrow$
$\forall\ N\ N',\ e \vdash N \rightarrow N' : A \rightarrow$
$e \vdash App\_l\ B\ (Abs\_l\ A\ M)\ N \rightarrow lsubst\ N'\ M' : lsubst\ N\ B$

$|\ tposr\_conv : \forall\ e\ M\ N\ A,\ e \vdash M \rightarrow N : A \rightarrow$
$\forall\ B\ s,\ e \vdash A \rhd B : s \rightarrow$
$e \vdash M \rightarrow N : B$

$|\ tposr\_subset : \forall\ e\ A\ A',\ e \vdash A \rightarrow A' : set \rightarrow$
$\forall\ B\ B',\ (A :: e) \vdash B \rightarrow B' : prop \rightarrow$
$e \vdash Subset\_l\ A\ B \rightarrow Subset\_l\ A'\ B' : set$

$|\ tposr\_sum : \forall\ e\ A\ A'\ s1,\ e \vdash A \rightarrow A' : s1 \rightarrow$
$\forall\ B\ B'\ s2,\ (A :: e) \vdash B \rightarrow B' : s2 \rightarrow$
$\forall\ s3,\ sum\_sort\ s1\ s2\ s3 \rightarrow$
$e \vdash Sum\_l\ A\ B \rightarrow Sum\_l\ A'\ B' : s3$

$|\ tposr\_pair : \forall\ e\ A\ A'\ s1,\ e \vdash A \rightarrow A' : s1 \rightarrow$
$\forall\ B\ B'\ s2,\ (A :: e) \vdash B \rightarrow B' : s2 \rightarrow$
$\forall\ s3,\ sum\_sort\ s1\ s2\ s3 \rightarrow$

$\forall\ u\ u',\ e \vdash u \to u' : A \to$
$\forall\ v\ v',\ e \vdash v \to v' : lsubst\ u\ B \to$
$e \vdash Pair\_l\ (Sum\_l\ A\ B)\ u\ v \to Pair\_l\ (Sum\_l\ A'\ B')\ u'\ v' : Sum\_l\ A\ B$

$|\ tposr\_pi1 : \forall\ e\ A\ A'\ s1,\ e \vdash A \to A : s1 \to e \vdash A \rhd A' : s1 \to$
$\forall\ B\ B'\ s2,\ (A :: e) \vdash B \rhd B' : s2 \to$
$\forall\ s3,\ sum\_sort\ s1\ s2\ s3 \to$
$\forall\ t\ t',\ e \vdash t \to t' : Sum\_l\ A\ B \to$
$e \vdash Pi1\_l\ (Sum\_l\ A\ B)\ t \to Pi1\_l\ (Sum\_l\ A'\ B')\ t' : A$

$|\ tposr\_pi1\_red : \forall\ e\ A\ A'\ s1,\ e \vdash A \to A' : s1 \to$
$\forall\ B\ B'\ s2,\ (A :: e) \vdash B \to B' : s2 \to$
$\forall\ s3,\ sum\_sort\ s1\ s2\ s3 \to$
$\forall\ u\ u'\ v\ v',\ e \vdash Pair\_l\ (Sum\_l\ A\ B)\ u\ v \to Pair\_l\ (Sum\_l\ A'\ B')\ u'\ v' : Sum\_l\ A\ B \to$
$\forall\ A'',\ e \vdash A'' \to A'' : s1 \to e \vdash A'' \rhd A : s1 \to$
$\forall\ B'',\ A'' :: e \vdash B'' \rhd B : s2 \to$
$e \vdash Sum\_l\ A''\ B'' \rhd Sum\_l\ A\ B : s3 \to$
$e \vdash Pi1\_l\ (Sum\_l\ A''\ B'')\ (Pair\_l\ (Sum\_l\ A\ B)\ u\ v) \to u' : A''$

$|\ tposr\_pi2 : \forall\ e\ A\ A'\ s1,\ e \vdash A \to A : s1 \to e \vdash A \rhd A' : s1 \to$
$\forall\ B\ B'\ s2,\ (A :: e) \vdash B \rhd B' : s2 \to$
$\forall\ s3,\ sum\_sort\ s1\ s2\ s3 \to$
$\forall\ t\ t',\ e \vdash t \to t' : Sum\_l\ A\ B \to$
$e \vdash Pi2\_l\ (Sum\_l\ A\ B)\ t \to Pi2\_l\ (Sum\_l\ A'\ B')\ t' : lsubst\ (Pi1\_l\ (Sum\_l\ A\ B)\ t)\ B$

$|\ tposr\_pi2\_red : \forall\ e\ A\ A'\ s1,\ e \vdash A \to A' : s1 \to$
$\forall\ B\ B'\ s2,\ (A :: e) \vdash B \to B' : s2 \to$
$\forall\ s3,\ sum\_sort\ s1\ s2\ s3 \to$
$\forall\ u\ u'\ v\ v',$
$e \vdash Pair\_l\ (Sum\_l\ A\ B)\ u\ v \to Pair\_l\ (Sum\_l\ A'\ B')\ u'\ v' : Sum\_l\ A\ B \to$
$\forall\ A'',\ e \vdash A'' \to A'' : s1 \to e \vdash A'' \rhd A : s1 \to$
$\forall\ B'',\ A'' :: e \vdash B'' \rhd B : s2 \to$
$e \vdash Sum\_l\ A''\ B'' \rhd Sum\_l\ A\ B : s3 \to$
$e \vdash Pi2\_l\ (Sum\_l\ A''\ B'')\ (Pair\_l\ (Sum\_l\ A\ B)\ u\ v) \to v' : lsubst\ (Pi1\_l\ (Sum\_l\ A''\ B'')\ (Pair\_l$
$(Sum\_l\ A\ B)\ u\ v))\ B''$

$where\ "G \vdash T \to U : s" := (tposr\ G\ T\ U\ s)$

$with\ tposr\_eq : lenv \to lterm \to lterm \to sort \to Prop :=$
$|\ tposr\_eq\_tposr : \forall\ e\ X\ Y\ s,\ e \vdash X \to Y : s \to e \vdash X \simeq Y : s$
$|\ tposr\_eq\_sym : \forall\ e\ X\ Y\ s,\ e \vdash X \simeq Y : s \to e \vdash Y \simeq X : s$
$|\ tposr\_eq\_trans : \forall\ e\ W\ X\ Y\ s,\ e \vdash W \simeq X : s \to e \vdash X \simeq Y : s \to e \vdash W \simeq Y : s$

$where\ "G \vdash T \simeq U : s" := (tposr\_eq\ G\ T\ U\ s)$

with tposr_coerce : lenv → lterm → lterm → sort → Prop :=
| tposr_coerce_conv : ∀ e A B s, e ⊢ A ≃ B : s → e ⊢ A ◃ B : s

| tposr_coerce_prod : ∀ e A B A' B',
∀ s, e ⊢ A' ◃ A : s →
e ⊢ A' → A' : s → e ⊢ A → A : s →
∀ s', (A' :: e) ⊢ B ◃ B' : s' →
A :: e ⊢ B → B : s' → A' :: e ⊢ B' → B' : s' →
e ⊢ (Prod_l A B) ◃ (Prod_l A' B') : s'

| tposr_coerce_sum : ∀ e A B A' B',
∀ s, e ⊢ A ◃ A' : s →
e ⊢ A' → A' : s → e ⊢ A → A : s →
∀ s', (A :: e) ⊢ B ◃ B' : s' →
A :: e ⊢ B → B : s' → A' :: e ⊢ B' → B' : s' →
∀ s", sum_sort s s' s" →
e ⊢ (Sum_l A B) ◃ (Sum_l A' B') : s"

| tposr_coerce_sub_l : ∀ e U P U',
e ⊢ U ◃ U' : set →
e ⊢ U → U : set → e ⊢ U' → U' : set →
U :: e ⊢ P → P : prop →
e ⊢ Subset_l U P ◃ U' : set

| tposr_coerce_sub_r : ∀ e U U' P,
e ⊢ U ◃ U' : set →
e ⊢ U → U : set → e ⊢ U' → U' : set →
U' :: e ⊢ P → P : prop →
e ⊢ U ◃ (Subset_l U' P) : set

| tposr_coerce_sym : ∀ e U V s, e ⊢ U ◃ V : s → e ⊢ V ◃ U : s

| tposr_coerce_trans : ∀ e A B C s,
e ⊢ A ◃ B : s → e ⊢ B ◃ C : s → e ⊢ A ◃ C : s

where " G ⊢ T ◃ U : s" := (tposr_coerce G T U s).

Lemma *wf_tposr : ∀ e M N T, e ⊢ M → N : T → tposr_wf e.*

Reserved Notation " G ⊢ M →⁺ N : B" (at level 70, M, N, B at next level).

Inductive *tposrp : lenv → lterm → lterm → lterm → Prop :=*
| tposrp_tposr : ∀ e X Y Z, e ⊢ X → Y : Z → e ⊢ X →⁺ Y : Z
| tposrp_trans : ∀ e W X Y Z, e ⊢ W →⁺ X : Z → e ⊢ X →⁺ Y : Z → e ⊢ W →⁺ Y : Z

where "$G \vdash M \to^+ N : B$" := (*tposrp* G M N B).

Definition *tposr_term* G M A := $\exists$ *M'*, $G \vdash M \to M' : A$.

Lemma *tposr_tposr_term* : $\forall$ G M *M'* A, *tposr* G M *M'* $A \to$ *tposr_term* G M A.

We add labels at applications and projections. Applications are annoted with the codomain of their function, for example, in $\mathrm{app}_{(x)B}(M\ N)$, we suppose that M can be typed with a product type of the form $\Pi x : \_.B$. Projections are annotated with their full domain, e.g in $\mathrm{pi}_{\Sigma x:A.B}(t)$, we suppose that t can be typed with the sum type $\Sigma x : A.B$. The reduction associated with labelled terms is the same as for unlabelled ones, plus reduction on labels. On the other hand, in the TPOSR relation, we allow coercion of type labels at will. Indeed an application of type B can be seen as an object of type B' if they are coercible.

We prove the basic metatheory of the system, including thinning (`Lambda.TPOSR.Thinning`), substituion (`Lambda.TPOSR.Substitution`) and substituion of TPOSR derivations (`Lambda.TPOSR.Substitutio`
Most proofs use mutual induction on the typing, well-formedness of contexts, equivalence relation and coercion relation derivations and are straightforward. However, much care must be taken in the staging of proofs. We denote by $\Gamma \vdash J$ any of the four judgements and use $\mathcal{R}$ to range over $\to$ (reduction), $\simeq$ (equivalence) and $\rhd$ (coercion). We were able to complete these proofs by following this order:

1. First, proving left reflexivity: If $\Gamma \vdash t \to t' : T$ then $\Gamma \vdash t \to t : T$
2. Then a preliminary form of context coercion: If $\Gamma, x : A, \Delta \vdash J$ and $\Gamma \vdash A \rhd B : s$ with $\Gamma \vdash A \to A : s$ and $\Gamma \vdash B \to B : s$ then $\Gamma, x : B, \Delta \vdash J$.
3. Then substitution: if $\Gamma, x : U, \Delta \vdash J$ and $\Gamma \vdash u \to u : U$ then $\Gamma, \Delta[u/x] \vdash J[u/x]$.
4. Then a preliminary form of substitution of TPOSR derivations: If $\Gamma, x : U, \Delta \vdash t\ \mathcal{R}\ t' : T$ and $\Gamma \vdash u \to u' : U$ with $\Gamma \vdash u' \to u' : U$ then $\Gamma, \Delta[u/x] \vdash t[u/x]\ \mathcal{R}\ t'[u'/x] : T[u/x]$.
5. Then we can prove right reflexivity: If $\Gamma \vdash t\ \mathcal{R}\ t' : T$ then $\Gamma \vdash t'\ \mathcal{R}\ t' : T$

After proving right reflexivity we can remove the side conditions in the statements of context coercion and substitution.

Once done with this we can prove generation lemmas for the TPOSR judgement. They are used to prove validity: if $\Gamma \vdash t \to t : T$ then $T = s$ for $s \in \mathcal{S}$ or there exists $s \in \mathcal{S}$ so that $G \vdash T \to T : s$. Once we have validity we just need to prove we have functionality of types to get uniqueness of types. Functionality says that types with equivalent components are equivalent. For example, functionality of pi is the property: if $\Gamma \vdash A \simeq B : s1$ and $\Gamma, x : A \vdash C \simeq D : s2$ then $\Gamma \vdash \Pi x : A.C \simeq \Pi x : B.D : s2$.

We can then prove uniqueness of types: $\Gamma \vdash t \to ? : T$ and $\Gamma \vdash t \to ? : U$ then $T = U =$ `Type` or there exists a sort s so that $\Gamma \vdash T \rhd U : s$. This is were labels are needed, indeed at the application case, in the original system we have the derivations:

$$\frac{\Gamma \vdash M : \Pi x : A.C \qquad \Gamma \vdash N : A}{\Gamma \vdash M\ N \to ? : C[N/x]}$$

$$\frac{\Gamma \vdash M : \Pi x : B.D \qquad \Gamma \vdash N : B}{\Gamma \vdash M\ N \to ? : D[N/x]}$$

By induction hypothesis we have $\Gamma \vdash \Pi x : A.C \rhd \Pi x : B.D : s$ for some s. However we cannot deduce yet that it implies $\Gamma, x : B \vdash C \rhd D : s$, as this requires injectivity of products which we can't prove, just like for the subject reduction proof. If we add labels we have:

$$\frac{\Gamma \vdash M : \Pi x : A.C \qquad \Gamma \vdash N : A}{\Gamma \vdash \mathrm{app}_{(x)E}(M\ N) \to ? : C[N/x]}$$

$$\frac{\Gamma \vdash M : \Pi x : B.D \qquad \Gamma \vdash N : B}{\Gamma \vdash \mathrm{app}_{(x)E}(M\ N) \to ? : D[N/x]}$$

Generation gives us the addidional hypothesis that $\Gamma \vdash E[N/x] \rhd C[N/x] : s$ and $\Gamma \vdash E[N/x] \rhd D[N/x]$, hence we are able to complete the proof. The same technique permits to prove uniqueness of types for projections.

Now that we have uniqueness of types, we can prove a Church-Rosser property for TPOSR: If $\Gamma \vdash t \to u : U$ and $\Gamma \vdash t \to v : V$ then there exists t' so that

$$\Gamma \vdash u \to t' : U$$
$$\Gamma \vdash u \to t' : V$$
$$\Gamma \vdash v \to t' : U$$
$$\Gamma \vdash v \to t' : V$$

The proof is by induction on the sum of the depth of the two derivations (`Lambda.TPOSR.ChurchRosserDepth`).

As a corrolary we get (`Lambda.TPOSR.ChurchRosser`): If $\Gamma \vdash t \simeq u : s$ then there exists x so that $\Gamma \vdash t \to^+_{\beta\pi} x : s$ and $\Gamma \vdash u \to^+_{\beta\pi} x : s$.

It is then easy to prove injectivity of products and sums using a similar argument as the one for untyped conversion : if $\Gamma \vdash \Pi x : A.C \simeq \Pi x : B.D : s$ then there exists x, $\Gamma \vdash \Pi x : A.C \to^+_{\beta\pi} x : s$ and $\Gamma \vdash \Pi x : B.D \to^+_{\beta\pi} x : s$. So we can derive that $x \equiv \Pi x : E.F$ for some E, F and there exists s' so that $\Gamma \vdash A \simeq E \simeq C : s1$, and $\Gamma, x : B \vdash C \simeq F \simeq D : s$.

The story doesn't ends here for injectivity as we have enriched the equivalence with a coercion system, so we need to prove a similar property for this system. Again this is not a trivial matter because we have a transitivity rule in this system which causes the same problem as for the usual equivalence. What we need to do is prove elimination of transitivity for the coercion system. The proof is again by induction on the depths of the two coercion derivations from which we build one (`Lambda.TPOSR.CoercionDepth`). Then we are able to prove injectivity of products and sums with respect to coercion (`Lambda.TPOSR.Injectivity`): If $\Gamma \vdash \Pi x : A.C \rhd \Pi x : B.D : s$ then there exists s' so that $\Gamma \vdash B \rhd C : s'$ and $\Gamma, x : B \vdash C \rhd D : s$.

Once we have injectivity for our type constructors we can easily prove subject reduction for the TPOSR system.

Translation to and from RUSSELL

To prove the equivalence of the TPOSR system and the original one, we need to show properties on the unlabelling of terms (denoted by the $|\_|$ function) (`Lambda.TPOSR.Unlab`). In this module, apart from easy properties of commutation of substitution, lifting and unlabelling, we show that labelled reductions entail reductions on the unlabelled terms and vice-versa: If $|t'| \to_{\beta\pi} u$, then there exists u', $|u'| = u$ and $t' \to_{\beta\pi} u'$.

Then the easy way, from TPOSR to RUSSELL (`Lambda.Meta.TPOSR_JRussell`, `Lambda.Meta.TPOSR_Russell`), is proved by a simple induction on derivations: If $\Gamma \vdash t \to t' : T$ then $|\Gamma| \vdash |t| : |T|$ and $|\Gamma| \vdash |t'| : |T|$.

The other way needs another lemma: stated simply, we need to show that if two typable labelled terms have syntaxically equal translates, then they have a common reduct and coercible types (`Lambda.TPOSR.UnlabConv`). Once armed with this result and its extension to contexts we can prove that unlabelling is complete with respect to the original system (`Lambda.Meta.Russell_TPOSR`), that is: If $\Gamma \vdash t : T$ then there exists Γ', t' and T' so that $|\Gamma'| = \Gamma$, $|t'| = t$ and $T' = |T|$, with $|\Gamma'| \vdash |t'| : |T'|$.

It is then possible to prove subject reduction for the original system: If $\Gamma \vdash t : T$ and $t \to_{\beta\pi} u$ then $\Gamma \vdash u : T$ (`Lambda.Meta.SubjectReduction`).

First we use the last lemma to go in TPOSR, then we translate the unlabelled reduction ta labelled one: $t' \to_{\beta\pi} u'$ with $|u'| = u$. We need just apply subject reduction for TPOSR and go back to RUSSELL.

References

[1] ADAMS, R. Pure Type Systems with Judgemental Equality. *Journal of Functional Programming 16* (2006), 219–246. `http://www.cs.rhul.ac.uk/~robin/ptseq8.ps.gz`.

[2] BARRAS, B. Coq en coq. Rapport de Recherche 3026, INRIA, Oct. 1996.

[3] SOZEAU, M. Russell Metatheoretic Study in Coq, experimental development, 2006. `http://www.lri.fr/~sozeau/research/russell.en.html`.